匆匆翻译没有校正，错误难免，阅读时请以英语原文为准，此翻译仅作参考用-----------郭叶军

用户手册

RoboCup Soccer Server

V7.06

1. 介绍

我们现在处于RoboCup的初级阶段，还有半个世纪的路要走，那时我们可以“建立一支仿人机器足球比赛队伍，并且战胜人类的足球世界冠军队”。这个目标带来的挑战是巨大的，而且吸引了全世界数以百记的研究者和他们的学生长年的投身于RoboCup。RoboCup是作为教育目标的和应用领域相平行的一个研究挑战，也用来刺激公共对机器人技术的兴趣和人工智能的发展。从1997年开始的每一年，世界各地的研究者聚集讨论，进行机器人的世界杯足球比赛。尽管机器人技术还是相当平常的，这个事情还是吸引了越来越多人的兴趣。

本指南的目的是引导仿真组队伍的最初工作，对熟练使用者是一本对server的参考指南。

1. 1背景

Mackworth 提出了进行机器人足球比赛的想法。不幸的是，这个想法并没有得到恰当的反响，直到Kitano, Asada, 和Kuniyoshi在日本提出名为Robot J_League的研究领域时，他们补充修改了这个想法。在1993年夏，几个美国研究者对Robot J_League发生了兴趣，于是就变成了Robot World Cup，简称为RoboCup。RoboCup有时也指RoboCup挑战或是RoboCup领域。

1995年，Kitano et al. [8]提出在1997年举行第一届机器人世界杯足球比赛和会议。RoboCup的目标是为AI和机器人技术提供一个新的标准问题，继深蓝后AI的又一个课题。RoboCup和AI中以前研究的问题不同，因为它致力于分布式解决方案，而不是集中式控制，而且研究的不仅仅是AI相关的传统领域，还包括以下范围：机器人技术，社会学，实时任务的关键系统，等等。

……

2.0 简介

2. 1开始

这个部分快速的介绍了RoboCup仿真组的主要部分。对于介绍的每一部分，你都会在本手册的以后内容得到详细的信息（如：参数配置，运行的选择项等等）。

2.1.1Server

Server是能使不同的队伍进行足球比赛的系统。因为比赛是以client/server方式进行的，所以对球队的开发编译没有任何限制。仅要求球队的开发工具提供通过UDP/IP连接的client/server支持。这是因为server和每个client之间的通讯都是通过UDP/IP端口实现的。每个client都是独立的进程，通过给定的端口和server连接。一支球队可以有最多11个client（或者说是球员）。当球员和server连接上后，所有的信息都通过这个端口传递。球员发送他们下一

步要做的动作请求给server（如踢球kick，转身turn，run等）。Server接收到这些消息后，执行请求，并相应的更新环境。另外，server向所有的球员提供感知sensory信息（如：关于足球，球门和其他球员的位置可视信息）。还有相当重要的一点，server是以离散的时间间隔（或周期）工作的实时系统。每个时间周期都有确定的分时，为了在某个周期执行，动作必须在正确的间隔到底server。因此，缓慢的反应会对球队的性能产生很大的影响，它会造成丢失执行动作的机会。

2.1.2Monitor

Monitor是一个可视化的工具，允许人们观看比赛时server到底发生了什么事情。在monitor上显示的信息包括比分,球队名字,所有球员和足球的位置。Monitor也提供了一个很简单的server接口。如：当两支球队都连接上后，在monitor上的“Kick－Off”按钮允许人类裁判开始比赛。正如你将发现的，在server上进行比赛，monitor并不是必需的。然而，如果有需要的话，可以同时和server连上很多的monitor（如你想在不同的终端显示同一场比赛）。

2.1.3Logplayer

Logplayer可以被看成一个录像球员。是用来进行重现比赛的工具。运行server时，可以加上一些选项，那么server就会将比赛的所有数据都存储在硬盘上。(很像在录像机上按下录制按钮)。然后，和monitor连接在一起的logplayer就可被用来重现比赛，不管重现多少次。这在进行球队的分析或者发现球队的强处和弱点时是很有用的。和录像球员差不多，logplayer也有开始play，停止stop，快进fast forwar和后退rewind按钮。而且logplayer也允许你跳到比赛的任一个周期（如你仅仅希望看到进球部分）。

2.2比赛规则

在比赛中，由server中的自动裁判和人类裁判来共同执行大量的规则。这节就是解释这些规则是如何工作，它们是怎么影响比赛的。

2.2.1由自动裁判裁决的规则

开球Kick－Off

在kick off之前（比赛开始前，和进球后），所有的球员都必须在她自己的半场。为了能够达到这样，在每次进球得分后，采取把比赛挂起5s时间。在这个间隔中，球员可以使用move命令移动到某点，而不是跑向这个点，这样会既费时又费体力stamina。如果在5s结束后，球员还呆在对方的半场，裁判会把她移到她们自己半场的随机位置。

进球Goal

如果一个球队进球得分，裁判要作很多事情。首先，她向所有的球员广播进球信息。然后更新比分，将球移到中点，并把比赛模式置为kick_off_x（x是left或right，代表左半场球队或右半场球队）。最后，她将比赛挂起5s，在此期间球员回到自己的半场（如在“开球Kick－Off”节所述）。

出界Out of Field

当足球出界时，裁判把足球放到一个合适的位置（边线，角球区或球门区）并且把比赛模式置为界外球kick_in，角球coner_kick，或者球门球goal_kick。如果是角球的话，裁判将把足球放到场内正确的角球区坐标（1m，1m）处。

清场Player Clearance

当比赛模式是开球kick_off，界外球kick_in，或角球corner_kick时，防守队员移出以足球为圆心，9.15m为半径的圆形区域。被移出的球员随机放置在圆形区域的周围。如果比赛模式是越位offside，所有的进攻球员被移回到没有越位的位置。这种情况下的进攻球员是指在越位范围内的所有球员以及位于以足球为圆心9.15m为半径的圆形区域内。如果比赛模式是球门球

goal_kick，所有的进攻球员会被移到罚球区外。当球门球发生时，进攻球员不能重新进入罚球区。当足球被踢出罚球区后，比赛模式马上被设置为正常play_on。

比赛模式控制Play-Mode Control
当比赛模式是开球kick_off，界外球kick_in，或者角球corner_kick时，裁判在足球因为kick命令产生移动时，马上把比赛模式设置为正常play_on。

中场时间和终场时间Half_Time and Time_Up
当上半场或下半场结束时，裁判暂时挂起比赛。每个半场的默认值是3000个仿真周期（大约5分钟)。如果在下半场结束后，还是平局的话，会进行加时赛。直到一方进球为止，就是被称为“ 金球 ” 规则或 “ 突然死亡法 ”。

2.2.2人类裁判裁决的规则
某些故意犯规动作，如故意阻挡等，很难由裁判(referee)自动判断，因为它与球员的意图有关。因此，server 为通过人来判断这些犯规动作提供了一种方式。人类裁判可以挂起比赛，赋予某个球队任意球。下面简单介绍一下此类犯规动作（在 RoboCup 2000 比赛时使用）：

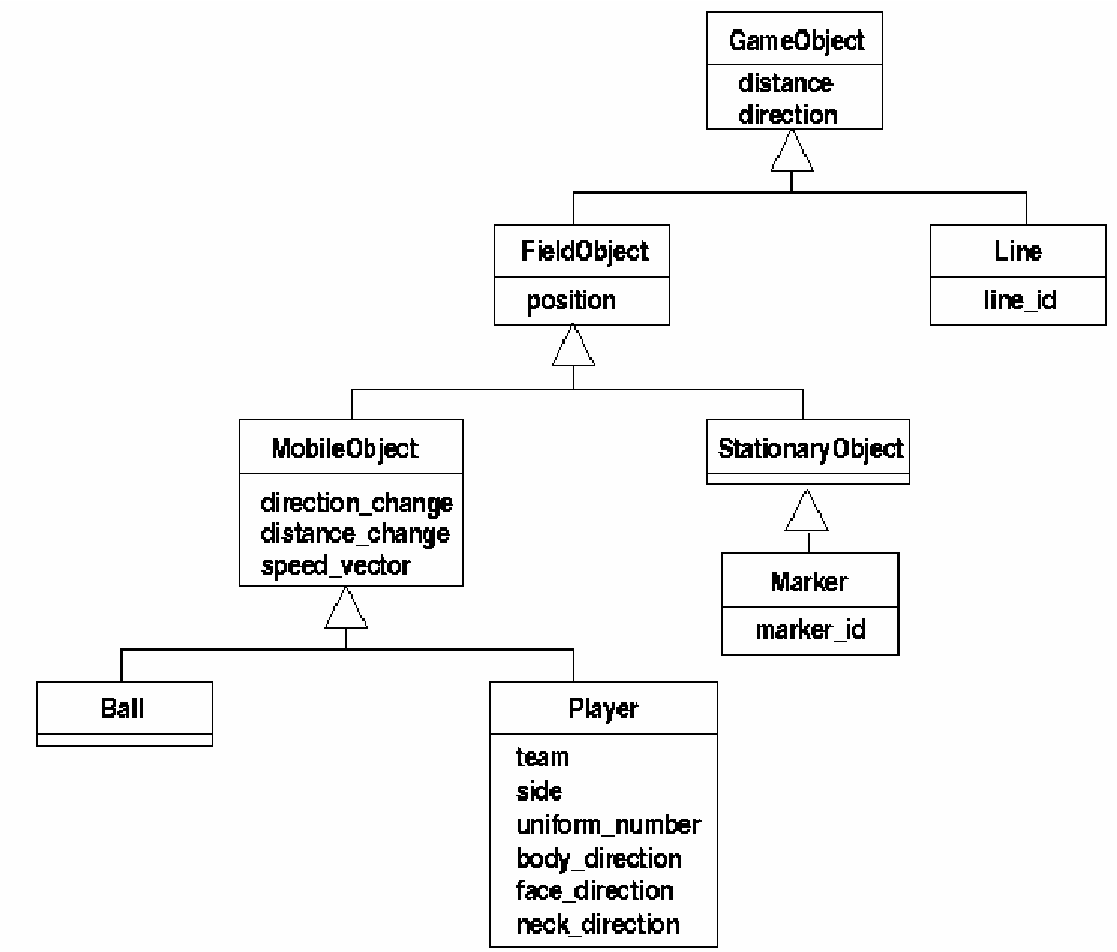
1 故意包围足球(Surrouding the ball)
2 故意用多名队员阻挡球(Blocking the goal with too many players)
3 故意长时间持球(Not putting the ball into play after a given number of cycles)
4 故意阻挡其他队员的移动(Intentionally blocking the movement of other players)
5 守门员滥用 catch 命令（守门员不允许在罚球区内重复使用 kick 和 catch 命令，因为这样为移动足球提供了安全的方式）
6 向 server 发送过多命令 Flooding the Server with Messages
每个 Client 在一个仿真周期内不能发送超过 3 或 4 个命令。过多的命令会使 server 阻塞。如果 server 发生阻塞，将在赛后进行检查。
7 不恰当的行为 Inappropriate Behaviour
如果显而易见的,一个球员以一种不恰当的方式干扰了比赛,那么人类裁判可以挂起比赛,并给对方球队一个任意球。

3.　开始
本节包括关于得到足球比赛服务端 Soccer Server 源码文件和安装这个软件的所有相关信息。下面讲述的步骤是在运行 GNU/Linux 2.2.17（用 uname 一sr 检查版本号）和 egcs 2.91.66（用 which g++检查版本号）的机器上测试通过的。当然如果进行了升级的话，也是可以的。在下面列出的命令中，假定一>是命令行提示符。

3.1 得到并安装 server
1.从下面的 Soccer Server 网站得到源码文件：
http://ci.etl.go.jp/~noda/soccer/server/ (Japan)
http://www.robolog.org/server/ (Germany)
。。。。。。

4. Soccer Server

4.1 对象 Objects



UML diagram of the objects in the simulation

4.2 协议

4.2.1client 命令协议

连接，重新连接，断开

From client to server	From server to client
(init *TeamName* [(version *VerNum*)] [(goalie)]) *TeamName* ::= $(-\|_\|a-z\|A-Z\|0-9)^{+}$ *VerNum* ::= the protocol version (e.g. 7.0)	(init *Side Unum PlayMode*) *Side* ::= l | r *Unum* ::= 1 ~ 11 *PlayMode* ::= one of play modes (error no_more_team_or_player_or_goalie)
(reconnect *TeamName Unum*) *TeamName* ::= $(-\|_\|a-z\|A-Z\|0-9)^{+}$	(init *Side Unum PlayMode*) *Side* ::= l | r *Unum* ::= 1 ~ 11 *PlayMode* ::= one of play modes (error no_more_team_or_player) (error reconnect)
(bye)	

如果 client 以大于等于 7.0 版本的协议成功的连接或重新连接，server 将额外发送以下的消息：一个包括 server 参数的消息，一个包括球员 player 参数的消息和一个包括球员类型的消息。格式如下所示。最后，球员将会接收到关于变换球员的消息（见 Sec.4.6）？？？

(server_param gwidth inertia_moment psize pdecay prand pweight pspeed_max paccel_max stamina_max stamina_inc recover_init recover_dthr recover_min recover_dec effort_init effort_dthr effort_min effort_dec effort_ithr effort_inc kick_rand team_actuator_noise prand_factor_l prand_factor_r kick_rand_factor_l kick_rand_factor_r bsize bdecay brand bweight bspeed_max

- (**server_param** *gwidth inertia_moment psize pdecay prand pweight pspeed_max paccel_max stamina_max stamina_inc recover_init recover_dthr recover_min recover_dec effort_init effort_dthr effort_min effort_dec effort_ithr effort_inc kick_rand team_actuator_noise prand_factor_l prand_factor_r kick_rand_factor_l kick_rand_factor_r bsize bdecay brand bweight bspeed_max baccel_max dprate kprate kmargin ctlradius ctlradius_width maxp minp maxm minm maxnm minnm maxn minn visangle visdist windir winforce winang winrand kickable_area catch_area_l catch_area_w catch_prob goalie_max_moves ckmargin offside_area win_no win_random say_cnt_max SayCoachMsgSize clang_win_size clang_define_win clang_meta_win clang_advice_win clang_info_win clang_mess_delay clang_mess_per_cycle half_time sim_st send_st recv_st sb_step lcm_st SayMsgSize hear_max hear_inc hear_decay cban_cycle slow_down_factor useoffside kickoffoffside offside_kick_margin audio_dist dist_qstep land_qstep dir_qstep dist_qstep_l dist_qstep_r land_qstep_l land_qstep_r dir_qstep_l dir_qstep_r CoachMode CwRMode old_hear sv_st start_goal_l start_goal_r fullstate_l fullstate_r drop_time*)
- (**player_param** *player_types subs_max pt_max player_speed_max_delta_min player_speed_max_delta_max stamina_inc_max_delta_factor player_decay_delta_min player_decay_delta_max inertia_moment_delta_factor dash_power_rate_delta_min dash_power_rate_delta_max player_size_delta_factor kickable_margin_delta_min kickable_margin_delta_max kick_rand_delta_factor extra_stamina_delta_min extra_stamina_delta_max effort_max_delta_factor effort_min_delta_factor*)

- for each available player type a message of the form
  (**player_type** *id player_speed_max stamina_inc_max player_decay inertia_moment dash_power_rate player_size kickable_margin kick_rand extra_stamina effort_max effort_min*)

client 控制

From client to server	Only once per cycle
(catch *Direction***)** *Direction* ::= **minmoment** ~ **maxmoment** degrees	Yes
(change_view *Width Quality***)** *Width* ::= narrow | normal | wide *Quality* ::= high | low	No
(dash *Power***)** *Power* ::= **minpower** ~ **maxpower** Note: backward dash consumes double stamina.	Yes
(kick *Power Direction***)** *Power* ::= **minpower** ~ **maxpower** *Direction* ::= **minmoment** ~ **maxmoment** degrees	Yes
(move *X Y***)** *X* ::= **-52.5** ~ **52.5** *Y* ::= **-34** ~ **34**	Yes
(say *Message***)** *Message* ::= a message	No
(sense_body) The server returns **(sense_body** *Time* **(view_mode {high | low} {narrow | normal | wide})** **(stamina** *Stamina Effort***)** **(speed** *AmountOfSpeed DirectionOfSpeed***)** **(head_angle** *HeadAngle***)** **(kick** *KickCount***)** **(dash** *DashCount***)** **(turn** *TurnCount***)** **(say** *SayCount***)** **(turn_neck** *TurnNeckCount***)** **(catch** *CatchCount***)** **(move** *MoveCount***)** **(change_view** *ChangeViewCount***))**	No
(score) The server returns **(score** *Time OurScore TheirScore***)**	No
(turn *Moment***)** *Moment* ::= **minmoment** ~ **maxmoment** degrees	Yes
(turn_neck *Angle***)** *Angle* ::= **minneckmoment** ~ **maxneckmoment** degrees turn_neck is relative to the direction of the body. Can be invoked in the same cycle as a turn, dash or kick.	Yes

server 可能会返回命令的出错信息：

(error unknown command)

(error illegal command form)

4.2.2client 感知协议

```
From server to client
(hear Time Sender "Message")
        Time ::= simulation cycle of the soccerserver
      Sender ::= online_coach_left | online_coach_right | referee | self | Direction
   Direction ::= -180 ~180 degrees
     Message ::= string
(see Time ObjInfo*)
        Time ::= simulation cycle of the soccerserver
     ObjInfo ::= (ObjName Distance Direction DistChange DirChange BodyFacingDir HeadFacingDir )
               | (ObjName Distance Direction DistChange DirChange
               | (ObjName Distance Direction)
               | (ObjName Direction)
     ObjName ::= (p ["Teamname" [UniformNumber [goalie]]])
               | (b)
               | (g [l|r])
               | (f c)
               | (f [l|c|r] [t|b])
               | (f p [l|r] [t|c|b])
               | (f g [l|r] [t|b])
               | (f [l|r|t|b] 0)
               | (f [t|b] [l|r] [10|20|30|40|50])
               | (f [l|r] [t|b] [10|20|30])
               | (l [l|r|t|b])
               | (B)
               | (F)
               | (G)
               | (P)
    Distance ::= positive real number
   Direction ::= -180 ~180 degrees
  DistChange ::= real number
   DirChange ::= real number
 HeadFaceDir ::= -180 ~180 degrees
 BodyFaceDir ::= -180 ~180 degrees
    Teamname ::= string
UniformNumber ::= 1 ~11
(sense_body  Time
             (view_mode {high | low} {narrow | normal | wide} )
             (stamina Stamina Effort)
             (speed AmountOfSpeed DirectionOfSpeed)
             (head_angle HeadAngle)
             (kick KickCount)
             (dash DashCount)
             (turn TurnCount)
             (say SayCount)
             (turn_neck TurnNeckCount)
             (catch CatchCount)
             (move MoveCount)
             (change_view ChangeViewCount))
             Time ::= simulation cycle of the soccerserver
          Stamina ::= positive real number
           Effort ::= positive real number
    AmountOfSpeed ::= positive real number
 DirectionOfSpeed ::= -180 ~180 degrees
        HeadAngle ::= -180 ~180 degrees
           *Count ::= positive integer
```

4.3 感知模型

每个 RoboCup 智能体有三种不同的感知。听觉感知检测裁判，教练和其他球员发送的消息。视觉感知检测场上的可视信息，象球员当前可视范围内的对象的距离和方向。视觉信息很象一个传感器，可以“ 见到 ”在球员身后的附近对象。自身 body 感知检测球员的当前物理状态，象它的体力 stamina，速度 speed 和头颈角度 neck angle。这三种感知联合给智能体提供了环境的一个较好的图景。

4.3.1 听觉感知模型

当球员或教练发送 say 命令时，server 就会发送听觉消息。裁判的调用也被作为听觉消息接收？。所有的消息都会被马上接收到。

从 server 发送的听觉消息的格式如下：

(hear Time Sender "Message")

Time：指示当前时间。
Sender：如果是其他球员发送的消息，那么是发送者的相当方向，否则就是下面的选项：
self：发送者是自己本人。
referee：裁判是发送者。
online_coach_left 或者 online_coach_ringt：发送者是在线教练。
Message 就是消息内容。最长 say_msg_size 个字节。裁判发送的消息的格式见 4.7.1 所示。影响听觉感知的 server 参数见表 4.1 所示：

Parameter in server.conf	Value
audio_cut_dist	50.0
hear_max	2
hear_inc	1
hear_decay	2
say_msg_size	512

Table 4.1.: Parameters for the aural sensor

听觉感知的能力
当球员的听力起码是hear_decay时，该球员才会听到一条消息。当球员听到一条消息后，听觉能力是以hear_decay下降的。每个周期球员的听力以hear_inc增长。听力的最大值是hear_max。为了避免球队阻塞信道使对手的交流失效，我们将球员的两支球队的听觉能力分离。现在的server.conf文件显示每个球员在每两个仿真周期内能从每支球队接收到一条消息a player can hear at most one message from each team every second simulation cycle.。
如果在同一时刻有更多的消息到达，那么球员真正接收到的消息是不确定的。(现在是根据消息到达的顺序来选择)。这条规则不包括裁判和球员自身发送的消息。换而言之，在一个时间周期内，一个球员能够发送一条消息同时接收其他球员的一条消息。

交流范围
仅仅在audio_cut_dist米的范围内，消息才能被传送到。如：在己方球门边的防守队员可以听到守门员发送的消息，但是在对方球门边的进攻球员是不可能听到这个消息的。裁判发送的消息能被所有的球员接收。

听觉感知例子？？？
This example should show which messages get through and how to calculated the hear capacity.
Example: Each coach sends a message every cycle. The referee send a message every cycle. The four players in the example all send a message every cycle. Show which messages gets through during 10 cycles (6 might be enough).

4.3.2视觉感知模型
视觉感知报告球员当前所能看到的对象。在每个sense_step（目前等于150ms）内，视觉信息被自动的发送到球员处。

从server发送的视觉信息遵循下面的基本格式：
(see ObjName Distance Direction DistChng DirChng BodyDir HeadDir)
其中

$$
\begin{aligned}
ObjName \quad ::= \quad & (\mathbf{p}\ \text{"}Teamname\text{"}\ UniformNumber\ \mathbf{goalie}) \\
& |\ (\mathbf{g}\ [\mathbf{l}|\mathbf{r}]) \\
& |\ (\mathbf{b}) \\
& |\ (\mathbf{f}\ \mathbf{c}) \\
& |\ (\mathbf{f}\ [\mathbf{l}|\mathbf{c}|\mathbf{r}]\ [\mathbf{t}|\mathbf{b}]) \\
& |\ (\mathbf{f}\ \mathbf{p}\ [\mathbf{l}|\mathbf{r}]\ [\mathbf{t}|\mathbf{c}|\mathbf{b}]) \\
& |\ (\mathbf{f}\ \mathbf{g}\ [\mathbf{l}|\mathbf{r}]\ [\mathbf{t}|\mathbf{b}]) \\
& |\ (\mathbf{f}\ [\mathbf{l}|\mathbf{r}|\mathbf{t}|\mathbf{b}]\ \mathbf{0}) \\
& |\ (\mathbf{f}\ [\mathbf{t}|\mathbf{b}]\ [\mathbf{l}|\mathbf{r}]\ [\mathbf{10}|\mathbf{20}|\mathbf{30}|\mathbf{40}|\mathbf{50}]) \\
& |\ (\mathbf{f}\ [\mathbf{l}|\mathbf{r}]\ [\mathbf{t}|\mathbf{b}]\ [\mathbf{10}|\mathbf{20}|\mathbf{30}]) \\
& |\ (\mathbf{l}\ [\mathbf{l}|\mathbf{r}|\mathbf{t}|\mathbf{b}])
\end{aligned}
$$

Distance，Direction，DistChng和Dirchng按如下方式计算出来的：

$$p_{rx} = p_{xt} - p_{xo} \tag{4.1}$$

$$p_{ry} = p_{yt} - p_{yo} \tag{4.2}$$

$$v_{rx} = v_{xt} - v_{xo} \tag{4.3}$$

$$v_{ry} = v_{yt} - v_{yo} \tag{4.4}$$

$$Distance = \sqrt{p_{rx}^2 + p_{ry}^2} \tag{4.5}$$

$$Direction = \arctan\left(p_{ry}/p_{rx}\right) - a_o \tag{4.6}$$

$$e_{rx} = p_{rx}/Distance \tag{4.7}$$

$$e_{ry} = p_{ry}/Distance \tag{4.8}$$

$$DistChng = (v_{rx} * e_{rx}) + (v_{ry} * e_{ry}) \tag{4.9}$$

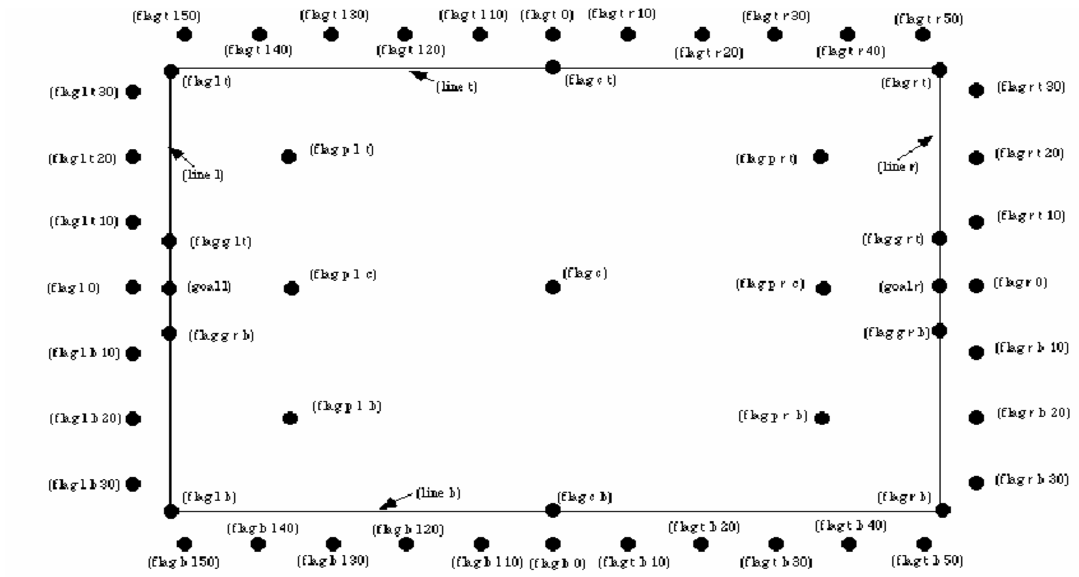
$$DirChng = [(-(v_{rx} * e_{ry}) + (v_{ry} * e_{rx}))/Distance] * (180/\pi) \tag{4.10}$$

$$BodyDir = PlayerBodyDir - AgentBodyDir - AgentHeadDir \tag{4.11}$$

$$HeadDir = PlayerHeadDir - AgentBodyDir - AgentHeadDir \tag{4.12}$$

其中(P_{xt}, P_{yt})是目标的绝对位置坐标，(P_{x0}, P_{y0})是接收视觉信息的队员自己本身的绝对坐标，(v_{xt}, v_{yt})是目标的绝对速度，(v_{x0}, v_{y0})队员自己的绝对速度。a_0是队员所面向的绝对方向。球员面向的绝对方向是球员的自身角度 BodyDir 和头部角度 HeadDir 的之和。另外(P_{rx}, P_{ry})和(v_{rx}, v_{ry})表示目标的相对位置和相对速度。(e_{rx}, e_{ry})表示平行于相对位置向量的单位向量。如果被观察者是球员的话，才会有 BodyDir 和 HeadDir，分别是被观察球员相对观察者的身体和头部的相对角度。如果两个球员的身体都是相同的角度，那么 BodyDir 就等于零。HeadDir 也一样。

对象（goal r）表示右半场球门线的中点。（f c）是场上中心的一个虚拟标记。。。。。。。。。



对于对象（l …），Distance 是视角的平分线和边线的交点到球员的距离。Direction 则是到边线的方向。？

视野范围

球员的可视部分依赖于几个因素。首先是 server 上的参数 sense_step 和 visible_angle，决定了视觉信息的时间间隔，以及球员正常视角时的角度。现在使用的是 150ms 和 90°。

球员通过改变 ViewWidth 和 ViewQuality 也可以改变视觉信息的频率和质量。

view_frequency 和 view_angle 由 4.13 式和 4.14 式计算得出。

view_frequency = sense_step * view_quality_factor * view_width_factor　　(4.13)

这里若 ViewQuality 是 high，那么 view_quality_factor 等于 1；若 ViewQuality 是 low，那么 view_quality_factor 等于 0.5。如果 ViewWidth 是 narrow，那么 view_width_factor 等于 2；如果 ViewWidth 是 normal，那么 view_width_factor 等于 1，如果 ViewWidth 是 wide，那么就等于 0.5。（质量越高，视角越小，发送时间间隔越大）

view_angle = visible_angle * view_width_factor　　(4.14)

这里若 ViewWidth 是 narrow，那么 view_width_factor 等于 0.5，若 ViewWidth 是 norrow，则 view_width_factor 等于 1，若 ViewWidth 是 wide，则 view_width_factor 等于 2。

球员能够“看到”在他领域内的对象（以visible_distance为半径的圆周）。如果某一对象在此范围内，但是不在球员的视野范围内，那么球员只知道对象的类型（足球，球员，球门或者是标记），而不知道对象的确切名字。就是说使用“B”，“P”，“G” 和“F”来作为对象的名字，而不是使用“b”，“p”,“g”和“f”。

图4.3表示了view_angle的意思。图中的观察球员由两个半圆组成，空心的半圆表示球员的前部。全黑的圆周代表场上的对象。只有在view_angle°/2的角度范围内，而且在visible_distance的距离范围内的对象才能被看到。因此，对象b和g是不可见的，其他的对象都是可见的。

对象f在观察者的正前方，它的角度被认为是0°。对象e会被认为是－40°，而对象d则会被认为是大约20°。

请参考图 4.3 的标志，球员所获得的视觉信息和距离的相关程度很大。对于近距离球员，它既可以看到它所属的球队同时也可以看到他的球员号码。然而，随着距离的增加，首先消失的是球员的号码。然后距离的增加还会导致球员的队别也分不清楚了。在服务器端假定这些距离是

$$\text{unum_far_length} \le \text{unum_too_far_length} \le \text{team_far_length} \le \text{team_too_far_length}$$

这里假定 dist 是球员和自己的距离，那么：

- 如果 $\text{dist} \le unum_far_length$,那么球员号码和球队名称都可见
- 如果 $\text{unum_far_length} < \text{dist} \le unum_too_far_length$，那么队名是可见的，但是队员号码有一定的概率看不到。这个概率根据 dist 是线性的从 1 到 0 减少的
- 如果 $\text{dist} \ge unum_too_far_length$,那么球员号码是不可见的。
- 如果 $\text{team_far_length} < \text{dist} \le team_too_far_length$，那么队名也存在一定的概率不可见，概率是随着 dist 的减少从 1 到 0 线性的递减的。
- 如果 $\text{dist} \ge \text{team_too}_far_length$,那么队名是不可见的

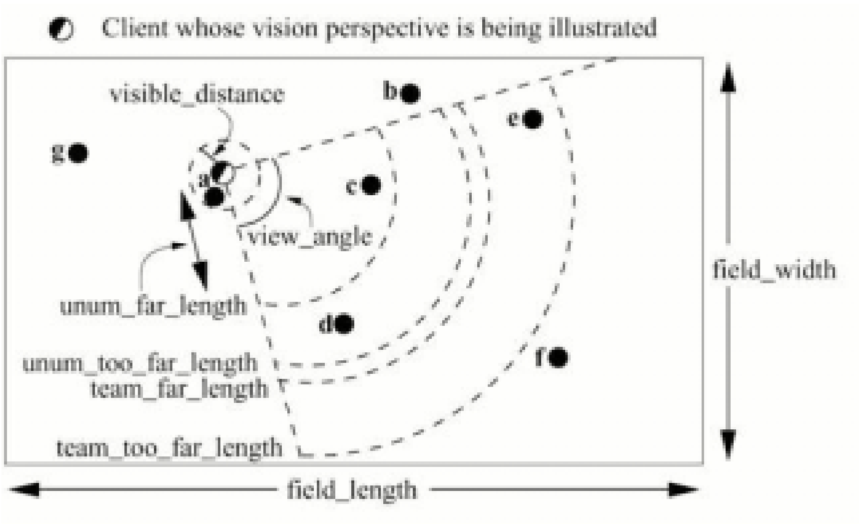


图 4.3

举例说明：根据图 4.3，假想根据所有的球员周围的环，对于球员 c 可以识别球队名称和号码，球员 d 可以识别除队名，而且有大概 50%的机会识别出号码；球员 e 则有 25%的机会识别出队名。对于球员 f 恐怕只能识别为一个匿名的球员了。

Parameter in server.conf	Value
sense_step	150
visible_angle	90.0
visible_distance	3.0
unum_far_length[a]	20.0
unum_too_far_length[a]	40.0
team_far_length[a]	40.0
team_too_far_length[a]	60.0
quantize_step	0.1
quantize_step_l	0.01

[a]Not in server.conf, but compiled into the server

Table 4.2.: Parameters for the visual sensors

视觉感知噪声模型

为了在视觉感知数据中引入噪声，server 发送的信息被进行了量化处理。如：无论远处的目标是球还是球员，目标的距离值按如下方式进行量化：

$$d^{'} = Quantize(\exp(Quantize(\log(d), quantize_step)), 0.1)$$

其中 $d, d^{'}$ 分别表示精确距离和相应的量化距离。且：

$$Quantize(V, Q) = ceiling（V/Q）*Q$$

这表示队员是不能知道远处物体的精确位置的。例如：当距离为 100.0 时，最大噪声可达到 10.0，但当距离在 10.0 之内时，噪声小于 1.0。

对于远处目标是旗或线的情况，距离值按如下公式量化：

$$d^{'} = Quantize(\exp(Quantize(\log(d), quantize_stepp_l)), 0.1)$$

4.3.3自身感知模型Body Sensor Model

自身感知返回球员当前的物理状态。每隔sense_body_step（目前使用100ms），就会自动的向球员发送自身感知信息。

自身感知消息的格式如下：

```
(sense_body Time
        (view_mode ViewQuality ViewWidth)
        (stamina Stamina Effort)
        (speed AmountOfSpeed DirectionOfSpeed)
        (head_angle HeadDirection)
        (kick KickCount)
        (dash DashCount)
        (turn TurnCount)
        (say SayCount)
```

(turn_neck TurnNeckCount)
(catch CatchCount)
(move MoveCount)
(change_view ChangeViewCount))

ViewQuality的取值是high或low。
ViewWidth取值是narrow，normal，wide。
AmountOfSpeed是球员速度的近似值。
DirectionOfSpeed是球员速度的近似方向。
HeadDirection是球员头部的相对方向。
变量Count是由server执行的对应命令的总量。如：DashCount＝134说明球员其时已经执行了134次dash命令。

参数的意义在它们使用的场合会做出说明的。如ViewQuality和ViewWidth参数就在4.3.2中叙述。对自身感知有影响的server参数见表4.3。

Parameter in server.conf	Value
sense_body_step	100

Table 4.3.: Parameters for the body sensor

4.4运动模型

在仿真周期内，对象的移动按如下公式进行计算：

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t):accelerate \qquad (4.18)$$

$$(p_x^{t+1}, p_y^{t+1}) = (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1}):move$$

$$(v_x^{t+1}, v_y^{t+1}) = decay \times (u_x^{t+1}, u_y^{t+1}):decay \quad speed$$

$$(a_x^{t+1}, a_y^{t+1}) = (0,0):reset \quad acceleration$$

其中，(p_x^t, p_y^t) 和 (v_x^t, v_y^t) 分别表示 t 时刻物体的位置和速度。*Decay* 是一个参数，分别由 *ball-decay* 和 *player-decay* 控制。(a_x^t, a_y^t) 表示对象的加速度，可以通过 ***dash***（针对队员）和 ***kick***（针对足球）的 *Power* 参数计算得到：

$$(a_x^t, a_y^t) = Power \times power_rate \times (\cos(\theta^t), \sin(\theta^t))$$

其中 θ^t 表示对象在 t 时刻的前进方向，power_rate 就是 dash_power_rate 或者是 kick_power_rate（见 4.53.）。如果对象为球员，它的方向就是球员脸朝向的方向。

对于足球，其方向的计算方法是：

$$\theta_{ball}^t = \theta_{kicker}^t + Direction$$

其中 θ_{ball}^t 和 θ_{kicker}^t 表示球和踢球队员当前的方向，而 *Direction* 是 ***kick*** 命令中第二个参数。

4.4.1 运动噪声模型

为了反映出实际比赛中球及球员运动的不确定性，server 在球及球员的运动过程中以及一些命令的参数中加入了一定的干扰因素。

首先考虑运动，干扰是以如下方式加入的：

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t) + (\tilde{r}_{r\max}, \tilde{r}_{r\max})$$

$\tilde{r}_{r\max}$ 为属于$[-\max, \max]$的随机数。*rmax* 的定义为：

$$rmax = rand \cdot \left|(v_x^t, v_y^t)\right|$$

rand 是 *player-rand* 和 *ball-rand* 的参数。

命令中的 *Moment 和 Power* 参数的干扰为：（用 argument 表示）

$$\arg ument = (1 + \tilde{r}_{rand}) \cdot \arg ument$$

4.4.2冲撞模型

在每个仿真周期的末尾，如果有两个对象相撞，那么会把对象后移使其不再重叠。然后两者的速度都乘以－0.1。注意，足球有可能穿过球员，只要在周期的末尾，足球和球员没有冲撞就行了。

4.5动作模型

4.5.1Catch抓球模型

守门员是唯一能执行catch命令的球员。守门员可以从任何方向抓到足球，只要足球在可抓范围内，守门员在罚球区内，且比赛模式是“ play_on ”。如果守门员以δ角度去catch足球，那么可抓范围是长宽分别是catchable_area_l和catchable_area_w的矩形区域（见图4.4）。如果足球在这个矩形区域内，能够被抓到的可能性是catch_probability，在外面则不能被抓到。目前使用的关于catch的参数值见表4.4。

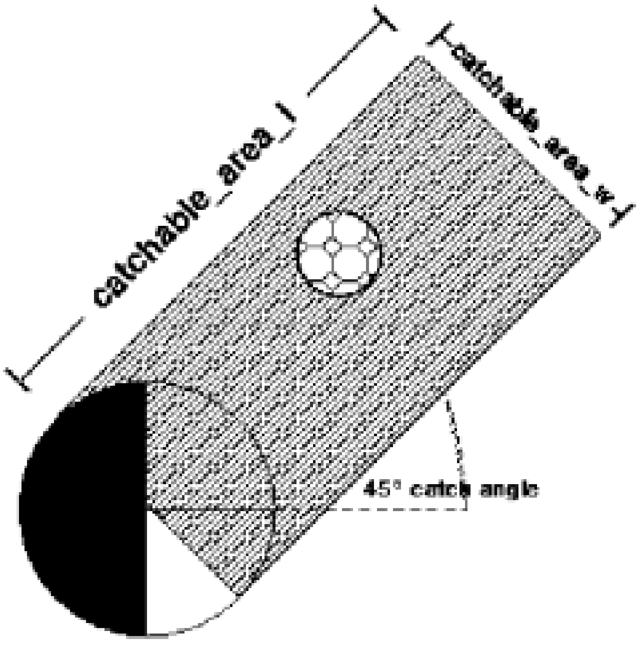


Figure 4.4.: Catchable area of the goalie when doing a catch 45

如果catch命令没有执行成功，那么要catch_ban_cycle后才能进行下一次的catch命令（在此期间的catch命令将不会发生任何效果）。如果守门员成功的抓球，比赛模式将在一个时间周期内先变为‘ goalie_catch_ball_[l|r] ’，再成为‘ free_kick_[l|r] ’。一旦守门员成功抓球，他就可以在罚球区范围内执行move命令。在kick踢球前，守门员最多可以执行goalie_max_moves次move命

令。更多的move命令将不起任何作用，同时server返回(error too many moves)。请注意，如果catch到足球，然后move，再kick足球很少的一段距离，接着马上重新catch….一旦move的次数超过了goalie_max_moves，会被认为是“ 非君子 ”的玩法。

Parameter in server.conf	Value
catchable_area_l	2.0
catchable_area_w	1.0
catch_probability	1.0
catch_ban_cycle	5
goalie_max_moves	2

Table 4.4.: Parameters for the goalie catch command

4.5.2Dash Model（包括stamina model）

Dash Model

命令dash给予球员一个加速度，和他身体朝向同向。命令dash的参数是power。值的范围见server.conf文件中的minpower和maxpower。目前dash模型使用的参数值见表4.5。

每个球员都有一定的体力stamina，会因为dash命令而消耗。在上下半场开始时，球员体力被置为stamina_max。如果球员向前加速（power>0），体力值降低power。如果是向后加速，体力值降低2倍power。如果dash需要的power超过了球员的体力值，那么power就会降低。异构队员在没有足够的体力值时可以使用额外体力。额外体力值依赖于球员类型和参数extra_stamina_deltra_min和extra_stamina_delta_max。

降低体力值后，server计算dash命令中power的有效部分。命令dash的有效部分edp（effective dash power）是由dash_power_rate和球员当前的effort决定的。球员的effort是介于effort_min和effort_max之间的值，受球员的体力管理决定（见下所述）。

$$edp = effort * dash_power_rate * power \quad (4.19)$$

edp和球员的当前身体方向被转化成向量，再加到球员当前的加速度$\overrightarrow{a_n}$（通常情况下是0，因为球员在一个周期内不能dash一次以上，而且球员也不可能不通过dash而是通过其他动作来获得加速度）。这是真的吗？（原文注）

从仿真周期n转换到仿真周期n＋1时，$\overrightarrow{a_n}$的使用如下：

1. $\overrightarrow{a_n}$被规格化到最大是**player_accel_max**的一个值。
2. $\overrightarrow{a_n}$被加到球员的当前速度$\overrightarrow{v_n}$。$\overrightarrow{v_n}$将被规格化到最大是**player_speed_max**的一个值。对于异构球员，最高速度介于**player_speed_max+player_speed_max_delta_min**和**player_speed_max+player_speed_max_delta_max** 之间（见player.conf文件）。
3. 噪声$\vec{n}$和风力$\vec{w}$也会影响$\overrightarrow{v_n}$。噪声和风力都在server.conf文件中定义。和风力有关的参数是**wind_force**，**wind_dir**和**wind_rand**。在目前的配置中，仿真环境没有加上风的影响。和噪声有关的参数是**player_rand**。噪声向量的方向和大小介于－$|\overrightarrow{v_n}|$ * player_rand和$|\overrightarrow{v_n}|$ * player_rand之间。

Basic Parameters server.conf		Parameters for heterogeneous Players player.conf		
Name	**Value**	**Name**	**Value**	**Range**
minpower	-100			
maxpower	100			
stamina_max	4000			
stamina_inc_max	45	*stamina_inc_max_delta_factor* *player_speed_max_delta_min* *player_speed_max_delta_max*	-100.0 0.0 0.2	25 — 45
extra_stamina[a]	0.0	*extra_stamina_delta_min* *extra_stamina_delta_max*	0.0 100.0	0.0 — 100.0
dash_power_rate	0.006	*dash_power_rate_delta_min* *dash_power_rate_delta_max*	0.0 0.002	0.006 — 0.008
effort_min	0.6	*effort_min_delta_factor* *extra_stamina_delta_min* *extra_stamina_delta_max*	-0.002 0.0 100.0	0.4 — 0.6
effort_max[a]	1.0	*effort_max_delta_factor* *extra_stamina_delta_min* *extra_stamina_delta_max*	-0.002 0.0 100.0	0.8 — 1.0
effort_dec_thr	0.3			
effort_dec	0.005			
effort_inc_thr	0.6			
effort_inc	0.01			
recover_dec_thr	0.3			
recover_dec	0.002			
recover_min	0.5			
player_accel_max	1.0			
player_speed_max	1.0	*player_speed_max_delta_min* *player_speed_max_delta_max*	0.0 0.2	1.0 — 1.2
player_rand	0.1			
wind_force	0.0			
wind_dir	0.0			
wind_rand	0.0			
player_decay	0.4	*player_decay_delta_min* *player_decay_delta_max*	0.0 0.2	0.4 — 0.6

[a]Not in `server.conf`, but compiled into the server

Table 4.5.: Dash and Stamina Model Parameters for Soccer Server 7

```
{if stamina is below recovery decrement threshold, recovery is reduced}
if stamina ≤ recover_dec_thr · stamina_max then
   if recovery > recover_min then
      recovery ← recovery − recover_dec
   end if
end if

{if stamina is below effort decrement threshold, effort is reduced}
if stamina ≤ effort_dec_thr · stamina_max then
   if effort > effort_min then
      effort ← effort − effort_dec
   end if
   effort ← max(effort, effort_min)
end if

{if stamina is above effort increment threshold, effort is increased}
if stamina ≥ effort_inc_thr · stamina_max then
   if effort < effort_max then
      effort ← effort + effort_inc
      effort ← min(effort, effort_max)
   end if
end if

{recover the stamina a bit}
stamina ← stamina + recovery · stamina_inc_max
stamina ← min(stamina, stamina_max)
```

Figure 4.5.: The stamina model algorithm

4. 球员的新位置$\overrightarrow{p_{n+1}}$是原先的位置$\overrightarrow{p_n}$加上速度$\overrightarrow{v_n}$（如：两个仿真周期间球员的最大距离改变是**player_speed_max**）。
5. **player_decay**被应用到球员的速度中：$\overrightarrow{v_{n+1}} = \overrightarrow{v_n} *$ player_decay。加速度$\overrightarrow{a_{n+1}}$被设为零。

体力模型

体力模型中有三个重要的参数：体力值stamina value，恢复recovery，效力effort。执行dash时会降低体力值，在每个周期体力值会有少量的增加。恢复recovery表明每个周期可以恢复多少体力。效力effort表明执行dash时的效力问题（见上节所述）。在文件server.conf和player.conf中涉及的体力模型的重要参数都是可变的，也见表4.5。基本上，图4.5中的等式表明在每个周期，如果体力值stamina低于某个极限，那么effort和recovery将会不断减少直至最少值为止。如果球员体力是某个极限之上，effort将会增大直到最大值。在每个半场开始，recovery都会被设为1.0，但是在比赛中它再也不会增加了。

4.5.3Kick Model

从server版本6到版本7，kick模型没有主要的变化，所以你以前的方法还是可用的。然而，因为server参数的改变，从某种角度上说，多次踢球将没有必要使用了。

命令**kick**有两个参数，踢球力量（介于**minpower**和**maxpower**之间）和球员踢球的角度。角度以度数表示，并介于**minmoment**和**maxmoment**之间（现在使用的参数值见表4.6）。

一旦命令**kick**到达server端，只要球员没有被置为越位，而且足球在可踢范围内，那么kick就会被执行。如果足球和球员之间的距离在0到kickable_margin之间，那么足球对这个球员来说就是可踢的。异构球员会有不同的可踢范围。本节我们讨论的距离是指两个对象（如足球和球员）的外圈间的距离，就是说我们讨论的距离是足球的中心到球员中心的距离减去足球的半径和球员的半径。首先要计算的是踢球的有效力量ep（effective kick power）：

$$ep = \text{kick power} * \text{kick_power_rate} \qquad (4.20)$$

如果足球不在球员的正前方，那么根据足球相当球员的位置，踢球有效力量将会减去某个数值。所以角度和距离都是很重要的。

如果足球相对球员身体角度是0°，就是说足球在球员面前，有效力量将不会改变。角度越大，有效力量将会减少的越多。最差的情况是足球在球员身后（180°）：有效力量将会减少25%。

有效力量的第二个重要变量是足球到球员间的距离：因为要执行kick命令，所以足球到球员的距离介于0到**kickable_margin**之间。如果距离是0，那么有效力量将不会再被减少。两者间的距离越大，有效力量减少的越多。如果距离等于**kickable_margin**，那么有效力量将会减少原来踢球力量的25%。

最坏的情况是足球在球员的后方最远处：将会使用50%的踢球力量。我们使用公式4.21来计算踢球有效力量（dir_diff是足球和球员身体的相对角度，dist_diff是足球和球员的相对距离）。

$$0 \le \text{dir_diff} \le 180^\circ \quad \wedge \quad 0 \le \text{dist_diff} \le \text{kickable_margin}:$$

$$ep = ep \cdot \left(1 - 0.25 \cdot \frac{\text{dir_diff}}{180^\circ} - 0.25 \cdot \frac{\text{dist_ball}}{\text{kickable_margin}}\right) \qquad (4.21)$$

踢球的有效力量被用来计算加速度 $\overrightarrow{a_{n_i}}$，这个加速度在周期n中加到足球的全局加速度 $\overrightarrow{a_n}$ 上（我们讨论的是多智能体系统，在同一个周期，可能有好几个球员踢球）。

有一个server参数**kick_rand**被用来在足球加速度中制造噪音。对默认球员，**kick_rand**等于0，将不会产生噪音。对于异构球员，**kick_rand**依赖于在文件player.conf中的参数

kick_rand_delta_factor以及实际的可踢范围。在RoboCup2000中，kick_rand was used to generate some noise during evaluation round for the normal players.??

在从仿真周期n到仿真周期n+1的转换中，加速度 $\overrightarrow{a_n}$ 被应用如下：

1. $\overrightarrow{a_n}$ 被规格化为最大是**baccel_max**的一个值。目前的server版本7中，最大的加速度等于踢球有效力量的最大值。
2. $\overrightarrow{a_n}$ 被加到足球的当前速度 $\overrightarrow{v_n}$ 。$\overrightarrow{v_n}$ 将被规格化到最大是**ball_speed_max**的一个值。
3. 噪声 $\vec{n}$ 和风力 $\vec{w}$ 也会影响 $\overrightarrow{v_n}$ 。噪声和风力都在server.conf文件中定义。和风力有关的参数是**wind_force**，**wind_dir**和**wind_rand**。和噪声有关的参数是**ball_rand**。噪声向量的方向和大小介于－$|\overrightarrow{v_n}|$ * ball_rand和$|\overrightarrow{v_n}|$ * ball_rand之间。

Basic Parameters server.conf		Parameters for heterogeneous Players player.conf		
Name	**Value**	**Name**	**Value**	**Range**
minpower	-100			
maxpower	100			
minmoment	-180			
maxmoment	180			
kickable_margin	0.7	*kickable_margin_delta_min* *kickable_margin_delta_max*	0.0 0.2	0.7 — 0.9
kick_power_rate	0.027			
kick_rand	0.0	*kick_rand_delta_factor* *kickable_margin_delta_min* *kickable_margin_delta_max*	0.5 0.0 0.2	0.0 — 0.1
ball_size	0.085			
ball_decay	0.94			
ball_rand	0.05			
ball_speed_max	2.7			
ball_accel_max	2.7			
wind_force	0.0			
wind_dir	0.0			
wind_rand	0.0			

Table 4.6.: Ball and Kick Model Parameters

4. 足球的新位置 $\overrightarrow{p_{n+1}}$ 是原先的位置 $\overrightarrow{p_n}$ 加上速度 $\overrightarrow{v_n}$（如：两个仿真周期间足球的最大距离改变是**ball_speed_max**）。
5. **ball_decay**被应用到足球的速度中：$\overrightarrow{v_{n+1}} = \overrightarrow{v_n}$ * player_decay。加速度 $\overrightarrow{a_{n+1}}$ 被设为零。

在当前的设置下，一次最佳踢球可以滚动距离为45。最佳踢球后53个周期，足球移动距离是43，其时速度小于0.1。而在15个周期内，则可以达到距离27－28，速度降为略大于1。踢球模型和现在的参数设置暗示复合踢球也是有用的，如：停住足球，将球踢到可踢范围内的一个更有利的点，然后踢往一个希望的方向。使用复合踢球的第二个可能是不用将足球移动相对位置（0，0°）处，而能将足球加速到最大速度值It would be another possibility to accelerate the ball to maximum speed without putting it to relative position (0,0°) using a compound kick.？

4.5.4Move Model

命令**move**可以把球员移动到场上的任何一个地方。只有在设置整个球队时**move**命令有效，在正常比赛期间是没有效果的。在上下半场开始前（比赛模式是‘before_kick_off’）以及进球后（比赛模式是‘goal_r_n’和‘goal_l_n’）才可以使用**move**命令。在这种情况下，只要比赛模式没有改变，球员可以被移动到自己半场的任何地点（就是说x<0），而且可以被移动任意多次。如果球员还在对方半场的话，那么将会被server移动到己方半场的随机位置。

使用**move**命令的第二个目的是当守门员成功抓到足球后，可以在罚球区内移动。（亦见4.5.1）。如果守门员成功抓到了球，就可以在罚球区内和足球一起移动。在他踢球前可以移动**goalie_max_moves**次。更多的**move**命令将不起任何作用，而且server返回(error too many moves)。

Parameter in server.conf	Value
goalie_max_moves	2

Table 4.7.: Parameter for the move command

4.5.5Say Model

球员可以使用**say**命令对其他球员进行广播消息。消息最长**say_msg_size**字节，有效字母是[-0-9a-zA-Z().+*/?<>_]（不包括方括号）。球员说的消息能够内**audio_cut_dist**距离内的任一支球队的球员听到（亦见4.3.1）。发到server的消息会被马上的发送到可听范围内的球员处。命令**say**的使用仅仅受球员听觉能力的限制。

Parameter in server.conf	Value
say_msg_size	512
audio_cut_dist	50
hear_max	2
hear_inc	1
hear_decay	2

Table 4.8.: Parameters for the say command

4.5.6 Turn Model

命令**dash**用来在球员的身体朝向上给予一个加速度，命令**turn**用来改变球员的身体朝向。命令**turn**的参数是moment；moment的有效值介于**minmoment**和**maxmoment**之间。如果球员是没有运动，那么他转过的角度将等于moment的值。然而，如果球员在运动，由于惯性的作用使其转身较为困难。球员真正转过的角度如下所示：

actual_angle = moment/(1.0 + inertia_moment * player_speed)　　　(4.22)

inertia_moment在文件server.conf中的默认值是5.0。因此（使用默认值），如果球员的速度是1.0，他能够做到的有效转身角度最大是±30。但是，因为球员不可能在同一个周期执行dash和turn命令，因此球员执行**turn**后最大速度是**player_speed_max** * **player_decay**，这就意味着默认球员的有效转身角度（使用默认值）是±60。

对于异构球员来说，inertia_moment是默认的**inertia_value**加上一个值，该值介于**player_decay_delta_min** * **inertia_moment_delta_factor** 和 **player_decay_delta_max** * **inertia_moment_delta_factor**之间。

Basic Parameters server.conf		Parameters for heterogeneous Players player.conf		
Name	Value	Name	Value	Range
minmoment	-180			
maxmoment	180			
inertia_moment	5.0	*player_decay_delta_min* *player_decay_delta_max* *inertia_moment_delta_factor*	0.0 0.2 25.0	5.0 10.0

Table 4.9.: Turn Model Parameters

4.5.7TurnNeck Model

使用**turn_neck**命令，从某种角度上，是在独立于球员身体的转动头颈。球员的头部角度是他的视野角度。命令**turn**改变球员的身体角度，而命令**turn_neck**则改变了球员相对他的身体的颈部角度。球员颈部的相对角度介于**minmoment**和**maxmoment**之间（在文件serever.conf中定义）。切记头颈角度是相对于球员身体的相对角度，如果球员执行了**turn**命令，而没有执行**turn_neck**命令，球员的视野角度也是会改变的。

在球员执行**turn**，**dash**，**kick**命令的同一个周期，也可以执行**turn_neck**命令。命令**turn_neck**不象**turn**那样受运动因素的影响。命令**turn_neck**的参数必须介于**minneckmoment**和**maxneckmomnet**之间。

Parameter in server.conf	Value
minneckang	-90
maxneckang	90
minneckmoment	-180
maxneckmoment	180

Table 4.10.: Parameter for the turn_neck command

表中的minneckang和maxneckang是指最后的头颈角度，见6.1.2所述（译者注）

4.6 异构球员

在server7.0版本中，引入了异构队员。Server在开始时产生**player_types**种类型以形成异构队员。基于权衡的原则，文件player.conf中定义了不同球员类型的不同能力。一场比赛中的球员使用相同的球员类型。类型0是默认类型，而且是同构的。

当球员和server连接上时，就会得到能够利用的球员类型信息（见4.2.1）。在‘ before_kick_off ’模式下，在线教练能够无限制的改变球员类型；在非‘ play_on ’的其他模式下，使用**change_player_type…**命令，能够改变球员类型**subs_max**次（见7.4）。

每当球员转换成其他类型，他的stamina，recovery和effort都复位到相应球员类型的初始（最大）值。

Parameter in player.conf	Value
player_types	7
subs_max	3

Table 4.11.: Parameter for substitutions and heterogeneous player types

4.7 Referee Model

自动裁判发送消息给球员，这样球员能够知道当前的比赛模式。自动裁判的规则和行为见2.2.1。球员用hear接收裁判消息。不管球员已经接收到其他球员的消息数量，他都能在每个仿真周期听到裁判的消息。

4.7.1 Play Modes and referee message比赛模式和裁判消息

比赛模式的改变由裁判宣布。裁判还有宣布其他的一些事情，象进球或者犯规。如果你读一下server的源代码，就会发现还有其他一些比赛模式现在没有使用。比赛模式和裁判消息通过

(referee String)宣布，String是比赛模式或者是消息字符串。比赛模式见表4.12，消息字符串见表4.13。

Play Mode	t_c	subsequent play mode	comment
'before_kick_off'	0	'kick_off_*Side*'	at the beginning of a half
'play_on'			during normal play
'time_over'			
'kick_off_*Side*'			announce start of play (after pressing the Kick Off button)
'kick_in_*Side*'			
'free_kick_*Side*'			
'corner_kick_*Side*'			
'goal_kick_*Side*'		'play_on'	play mode changes once the ball leaves the penalty area
'goal_*Side*'			currently unused (but see Tab. 4.13).
'drop_ball'	0	'play_on'	
'offside_*Side*'	30	'free_kick_*Side*'	for the opposite side

where *Side* is either the character 'l' or 'r', *OSide* means opponent's side.
t_c is the time (in number of cycles) until the subsequent play mode will be announced

Table 4.12.: Play Modes

Message	t_c	subsequent play mode	comment
goal_*Side*_*n*	50	'kick_off_*OSide*'	announce the *n*th goal for a team
foul_*Side*	0	'free_kick_*OSide*'	announce a foul
goalie_catch_ball_*Side*	0	'free_kick_*OSide*'	
time_up_without_a_team	0	'time_over'	sent if there was no opponent until the end of the second half
time_up	0	'time_over'	sent once the game is over (if the time is ≥ second half **and** the scores for each team are different)
half_time	0	'before_kick_off'	
time_extended	0	'before_kick_off'	

where *Side* is either the character 'l' or 'r', *OSide* means opponent's side.
t_c is the time (in number of cycles) until the subsequent play mode will be announced

Table 4.13.: Referee Messages

其中第三行的goalie_catch_ball_Side接下去应该是free_kick_Side，而不是free_kick_Oside？？？

4.8 足球仿真

在4.4中，我们描述了根据对象的速度和加速度的运动情况。在本节，我们讨论在仿真时，速度和加速度应用于哪个时刻。

4.8.1 仿真法则的描述

在server处，时间是以离散步骤进行更新的。每个仿真步骤是100ms。在每个仿真步骤之内，对象（就是球员和足球）停在原先的位置。如果球员在某个仿真步骤内想有所动作的话，那么此动作会在仿真周期向下一个周期转换时应用到球员和足球上。基于比赛模式，并不是允许球员

执行所有的动作（如：在‘ before_kick_off ’模式下，球员只能**turn**和**move**，而不能dash），因此只有允许的动作会被执行并产生作用。

如果在一个步骤内，有好几个球员踢球，那么所有的效用将被相加，最后得出一个加速度。如果这个加速度大于足球的最大加速度，那么就会被量化到它的最大值。移动了对象后，server检查冲撞，如果冲撞发生，就更新速度（亦见4.4.2）。

对对象应用加速度和速度时，它们的顺序是随机的。改变对象的位置，更新它们的速度、加速度后，自动裁判检查场上情况，改变比赛模式或者对象位置（如果有必要的话）。比赛模式的改变会被马上宣布。最后，更新每个球员的体力。

4.9 使用Soccerserver
4.9.1server参数（在文件server.conf中的可调整参数表）
见英文版56页，57,58页。

5． The Soccer Monitor（略）
有两个版本version 1和version 2

v1包括三种信息
1. showinfo_t: information needed to draw the scene
2. msginfo_t : contains the messages from the players and the referee shown in the bottom windows
3. drawinfo_t: information for monitor to draw circles, lines or points (not used by the server)

v2包括五种信息：
1. showinfo_t2: information needed to draw the scene
2. msginfo_t : contains the messages from the players and the referee shown in the bottom windows
3. player_type_t: information describes different player's ability
4. server_params_t: parameters and configurations of soccerserver
5. player_params_t: parameters of player

5.3.2 Monitor送到Server的信息

首先用下面的命令和server建立连接	(dispinit) | (dispinit version 2)
点击‘ kick_off ’按钮时，发送	(dispstart)
向server发送在（x，y）处side方犯规信息	(dispfoul x y side)
罚出场	(dispdiscard side unum)
将球员移到到(posx,posy)/SHOWINFO_SCALE，身体方向是ang ，在server7.02中新增	(dispplayer side unum posx posy ang)

其他略。

6. Soccer Client
6.1 协议
本节讨论client和server间的协议的基本情况。更多的细节见server小节。

注意初始化和重新连接命令将被送到运行server机器的球员UDP端口（默认是6000），得到响应后，该球员就和server分配的端口绑定，以后的信息也发送到这里。Server从这个端口发送

初始化响应信息（参见1.2.1）。所有发送到server和从server端接收的命令都是普通字符串和。。。All the commands sent to or received from the server are strings of common character and are in a pair of prantesis. ？？？？

6.1.1 初始化和重新连接

每个要和server连接的球员都要首先介绍自己。这就好像是次握手，在开始时进行一次，在你希望重新连接时随意进行。

初始化

球员要以下面的格式来发送init命令：

(init TeamName [(version VerNum)] [(goalie)])

守门员必须在init命令中包括“（goalie）”，这样server才会运行抓球或做其他守门员才能做的动作。请注意每个球员只能有一个守门员或者没有守门员（没有规定必须要有一个守门员）。

Server用如下格式的消息来响应你的初始化消息：

(init Side UniformNumber PlayMode)

或者是出错消息（如果出错的话，就是说你初始化了不止两支队伍，一支球队的人数超过了11个，或者一支球队中使用了不止一个守门员）：

(error no more team or player or goalie)

Side是你球队比赛时的标示，一个字母，**l**(left)或者是**r**(right)。UniformNumber是球员的球员号（球员通过它们的球员号被识别）。PlayMode是表示一个有效比赛模式的一个字符串。

如果你和server版本是7.00或更高的连接，你会接收到更多的server参数，球员参数和球员类型信息（最后两个是关于异构球员的特性）。精确的格式见附录A。

(server_param Parameters . . .)
(player_param Parameters . . .)
(player_type id Parameters . . .)

到此握手结束，你的球员被作为有效球员识别了。

重新连接

重新连接是很有用的，因为无需重启比赛就可以修改球员的程序。只能在无PlayOn的比赛模式中使用（如：在半场时间）。

使用下面的格式进行重新连接：

(reconnect TeamName UniformNumber)

你将会得到下面格式的响应：

(reconnect Side PlayMode)

如果在PlayOn比赛模式，则返回：

(can't reconnect)

如果因为某种错误，没有球员可以重新连接，则返回：

(error reconnect)

如果球队名字出错，也会返回：

(error no more team or player or goalie)

在这里，如果你是和server版本是7.00或更高的连接，也会返回更多的server参数，球员参数和球员类型信息。

断开连接

在你断开之前，你可以发送给server一个bye命令。这个命令将会把球员从场上移出。

(bye)

将不会用server的返回信息。

版本控制

由于server的不断开发，每年都有新的特性被加入，为了支持这些新特性，需要修改原先的协议。为了能够向下兼容老版本client程序，为了是开发者更方便的工作，故加入了协议版本控制Protocols Version Control。每个球员都要在**init**命令中说明他的通讯协议，这样server会以适当的格式发送消息。(本手册没有提到包括version的init命令的具体格式，译者注)。

但是注意即使通讯协议没有改变，如果仿真规则变化的话，也是会影响整场比赛的。

6.1.2 控制命令

所有球员的运动行为都是由几个命令组成的，这几个命令前面已经有所谈及。

这些命令的结果有些复杂，并且依赖于很多仿真因素。对每个命令的执行细节见serer小节。

(turn Moment)

Moment介于－180到180之间。这个命令将在球员当前身体角度的基础上转过Moment度角。

(dash Power)

这个命令给予球员一个加速度，和球员身体朝向同向（不一定就是当前的速度方向）。Power介于**minpowe**r和**maxpwer**之间，现在这两个值是－100，100。

(kick Power Direction)

在Direction方向用Power给足球加速。Direction是相对于球员当前身体方向的相对角度，power是介于**minpower**和**maxpower**之间。

(catch Direction)

守门员的特有命令：以Direction方向去试图抓球，direction是相对球员当前身体方向的相对角度。如果命令被成功执行，在守门员踢球kick前，足球将一直被守门员控制。

(move X Y)

这个命令仅能在开球前或是进球后执行。它将球员在一个周期内移动到绝对坐标（x，y）处，x介于－54到54之间，　y介于－32到32之间。在开球前，这是很有用的。(这个命令也可以在守门员catch到足球后，在罚球区内进行———译者注)。

注意，在一个仿真周期内，只能执行上述五个命令中的其中一个。(也就是说，如果球员在一个周期里面发送以超过一个命令，那么就会被随机执行其中的一个命令，通常情况下是先到达的那个)。

(turn_neck Angle)

这个命令可以和其他命令在一个周期内同时被发送和执行。头颈将在运行的角度基础上转过Angle角度。请注意最后的头颈角度将在**minneckanf**和**maxneckang**之间，两者的默认值分别是－90和90。头颈角度是相对球员身体角度的相对角度。

通讯命令

两个球员间进行通讯交流的唯一办法是使用**say**命令进行广播，使用**hear**感知听觉信息。

(say Message)

这条命令在场上广播消息，任何球员，只要他足够近（由audio_cut_dist确定，默认值是50.0米），还有足够的听觉能力，他就会听到这条消息。消息是有效字符的一个字符串。

(ok say)

命令被成功执行后的返回信息。

(error illegal_command_form)

如果有错误存在的话，将会从server返回如上格式的消息。

其他命令

还有另外两种形式的命令

- 数据请求命令

(sense body)

向server请求发送sense_body信息。请注意如果你和版本6.00或更高的server相连的话，server会在每个周期都向你发送sense_body信息。

(score)

向server请求发送比分信息。Server将会用以下的格式来响应：

(score Time OurScore OpponentScore)

- 模式改变命令

(change_view Width Quality)

改变球员的视觉参数。Width是**narrow**，**normal**或者**wide**。Quality是**high**或者是**low**。视觉感知返回的信息数量和质量取决于视觉宽度和质量的设定值。但是需要注意的是发送信息的频率也是取决于这些参数（如：如果你将quality从high改变到low，那么频率就会加倍，就是说两次视觉感知的时间间隔成为原先的一半）。

6.1.3 感知信息

感知信息被有规律的发送给所有的球员（如每个周期或者是每1.5个周期）。所以没有必要向server发送命令请求得到这些信息。

所有感知返回的信息都有时间戳（Time），表明该数据被发送时server端的周期数。这个时间是很有用的。

视觉感知

视觉感知是最重要的感知器，但是有一点复杂。这个感知返回球员能够见到的对象信息（就是说在视角范围，又不很远的对象）

信息的主要格式如下：

(see Time ObjInfo ObjInfo . . .)

ObjInfo是如下的格式：

(ObjName Distance Direction [DistChange DirChange [BodyFacingDir HeadFacingDir]])

或

(ObjName Direction)

注意返回的对象信息和他的距离有关。对象相距越远，得到的信息越少。更多的信息参见附录。

ObjName是下面的一种：

(p [TeamName [Unum]])

(b)
(f FlagInfo)
(g Side)

p表示球员，**b**表示足球，**f**表示标志，**g**表示球门。

Side是**l**表示左半场或者是**r**表示右半场。关于FlagInfo更多的信息参加附录。

听觉感知

听觉感知返回能在场上听到的消息。可能来自在线教练，裁判或者其他球员。

格式如下：

(hear Time Sender Message)

Sender是下面的一种：

Self：发送者是自己；

Referee：发送者是比赛裁判。

Online_coach_l或者是**online_coach_r**

Direction：如果是其他球员发送的消息，那么sender将会被发送者相对于自己的角度所代替。

自身感知

自身感知返回球员的所有状态，如体力，视觉模式，球员在每个周期刚开始时的速度：

(sense_body *Time* (view_mode { high | low } { narrow | normal | wide }) (stamina *Stamina Effort*) (speed *Speed Angle*) (head_angle *Angle*) (kick *Count*) (dash *Count*) (turn *Count*) (say *Count*) (turn_neck *Count*) (catch *Count*) (move *Count*) (change_view *Count*))

最后8个参数是接收到命令的计数值。使用这个计数值用来寻找丢失或延误的消息。

（下面的以后再翻）

接下去是对sampleclient的分析，以及simpleclient的基本。

第七章是coach，第八章是参考文献和以后方向，再接下去是有关命令的索引。

（注：文章提到的附录不知道是指什么。 用acrobat打开pdf文件得到的页数和原文标注的页数是不相同的）。

6.2 如何编写 clients

本节简要的描述了编写 client 的首要几个步骤。

6.2.1 Sample Client

server 发布版本包括了一个很简单的 client 程序，叫 sampleclient。在“ sampleclient ”的目录下，当你编译 server 时，sampleclient 也自动被编译。

Sampleclient 不是一个独立的 client：它仅仅是一个简单的“ 管道 ”，将标准输入命令（键盘）送到 server 端，将从 server 过来的信息显示在标准输出（屏幕）。因此当我们调用 sampleclient 时是不会有任何反应的。使用者要从键盘输入命令，读取显示在终端上的感知信息。（其实是很难读取感知信息的，因为 server 每秒种要发送大约 17 个感知信息。

为了理解 client 应该做什么，将会从 server 接收到什么，Sampleclient 是非常有用的。

如何使用 sampleclient

这里是 sampleclient 的典型使用方法。

1. 在 sampleclient 目录下调用 client

```
% ./client SERVERHOST
```

SERVERHOST 是运行 server 的 hostname

如果端口有变化，则使用：

```
% ./client SERVERHOST 6005
```

2. 从键盘输入初始化 init 命令

```
(init MYTEAMNAME (version 7))
```

MYTEAMNAME 是想使用的球队名字

然后场上就会出现一个球员。同时，程序开始将 server 发送来的感知信息显示到终端。这里是一个典型的输出：

```
send 6000 : (init foo (version 7))
recv 1567 : (init r 1 before_kick_off)
recv 1567 : (server_param 14.02 5 0.3 0.4 0.1 60 1 1 4000 45 0 0.3 0.5 ...
recv 1567 : (player_param 7 3 3 0 0.2 -100 0 0.2 25 0 0.002 -100 0 0.2 ...
recv 1567 : (player_type 0 1 45 0.4 5 0.006 0.3 0.7 0 0 1 0.6)
recv 1567 : (player_type 1 1.16432 28.5679 0.533438 8.33595 0.00733326 ...
recv 1567 : (player_type 2 1.19861 25.1387 0.437196 5.92991 0.00717675 ...
recv 1567 : (player_type 3 1.04904 40.0956 0.436023 5.90057 0.00631769 ..
recv 1567 : (player_type 4 1.1723 27.7704 0.568306 9.20764 0.00746072 ...
recv 1567 : (player_type 5 1.12561 32.4392 0.402203 5.05509 0.00621539 ..
recv 1567 : (player_type 6 1.02919 42.0812 0.581564 9.53909 0.00688457 ..
recv 1567 : (sense_body 0 (view_mode high normal) (stamina 4000 1) ...
recv 1567 : (see 0 ((g r) 61.6 37) ((f r t) 49.4 3) ((f p r t) 37 27) ...
recv 1567 : (sense_body 0 (view_mode high normal) (stamina 4000 1) ...
...
```

解释略

3. 输入 move 命令移动球员到初始位置。球员原先出现在场外。使用者需要用如下的命令将球员移到它的初始位置：

```
(move -10 10)
```

然后球员就会移到点（－10，10）。

刚才说道 client 的输出信息是无穷的，因此使用者不会看到他们输入的字符串。所以他们必须进行盲打输入。（注：也可以使用“ %client SERVERHOST > /dev/null ”，这样丢掉返回的感知信息。利于命令的输入）。

4. 点击 server 上的“ Kick-Off ”按钮，然后比赛就开始了。使用者会看到每个感知信息的时间数在不断的增加（是 see 和 sense_body 信息的第一个数字）。

5. 然后，使用者就可以输入任何正常命令 turn，dash，kick 等等。如：使用者可以输入下面的命令将球员转到右方。

```
(turn 90)
```

球员也可以全力向前冲：

```
(dash 100)
```

如果球员离足球足够近，他就可以用 power＝50 将球踢往左边：

```
(kick 50 -90)
```

再次提醒一下，因为感知输出是无穷的，所以必须进行盲打输入命令。

Sampleclient 的总体结构

1. 打开一个 UDP socker，和 server 端口连接。（init_connection()）
2. 进入一个 real－write 循环（message_loop），在此，有两个进程被平行执行。
   - 读取用户输入的命令（通常是从键盘输入），然后将其发送到 server（send_message( )）。
   - 接收从 server 过来的感知信息（receive_message()），并将其送到标准输出（通常是终端）。

为了实现平行执行，samplecliet 使用了 select()函数。该函数使在一个单独的进程中等候 socker 和 stream 的多重输入。当 select()被调用时，它一直等待，直到其中一个 socket 和 stream 得到输入数据，并分辨出是哪个 socket 或 stream。关于 select()的跟多用法，请参考 man 页或用户文档。

Sampleclient 中的一个重要提示是当他从 server 接收到信息后，必须修改 server 的端口号。因为当 server 接收到一个 init 命令后，它为这个申请的球员新建了一个端口。在“ lient.c ”中的下面部分实现（大约在 147 行）：

```
        printf( "recv %d : ", ntohs(serv_addr.sin_port));
+       sock->serv_addr.sin_port = serv_addr.sin_port ;
        buf[n] = '\0' ;
```

6.2.2 Simple Clients

为了开发一个完整的 clients，用户必须写出代码，实现在接收的感知信息的基础上产生需要进行的命令字符串。

当然这不是个简单的任务（所以很多研究者将 RoboCup 作为一个研究项目），有很多种方法可以实现。简单的说，为了开发 clients，用户需要了解下面的部分：

〔**Sensing**〕分析感知信息：正如以前章节叙述的，server 以 S-expression 方式发送不同的感知信息。因此，client 需要分析 S-expression。然后，client 进行分析信息得到一个内部描述。如：client 需要分析视觉信息来估计球员位置和场上状态，因为视觉信息仅包含了场上标记是移动对象的相对位置。

〔**Action Interval**〕控制发送命令的时间间隔：因为 server 每隔 100ms 接收一次身体命令（turn，dash，kick），client 在发送命令前需要等待恰当的时间间隔。

〔**Parallelism**〕并行的执行感知和动作进程：因为 server 异步的发送感知信息和命令，client 需要执行感知进程来出来感知信息，同时也要执行一个动作进程，平行的控制发送命令。

〔**Planning**〕球员规划：在使用感知信息的基础上，client 需要产生比赛的恰当的命令序列。当然，这就是开发 client 的最后目标！!

这里是“ 独立 ”stand－alone 球员的两个简单的例子，sclient1 和 sclient2，仅仅是追逐足球然后踢到对方的球门中。源码可以在这里找到：

ftp://ci.etl.go.jp/pub/soccer/client/noda-client-2.0.tar.gz

在这些例子中，上面列的部分的实现如下所述：

接下去是 sclient1 和 sclient2 的简要分析，无很大意义。略。

6.2.3 Tips

我们收集了一些开发 client 程序的小技巧和提示

- 调试是在开发过程中的最大问题，所以尽可能的找到简单的调试方法。
- 察看在某一个条件下，程序中的变量情况的一个有效的简单方法是使用 abort()命令或者一些断点，迫使程序产生 core－dump；然后使用 gbd 来调试产生的 core 文件。
- 记录发送给 server 以及从 server 接收的每一条消息。这样对调试是很有帮助的。

- 如果你是一个新手，那么使用现成的 socket 和分析问题的库函数是很有帮助的。
- 记住在发送 init 命令时，要带有版本号。虽然这是个可选项，但是默认值是 3.00，并不是我们希望的。
- 即使你的抓球能力是 1.00，也有可能不能成功抓球，因为返回的位置信息可能有错误。
- 你碰到的第一个难点将是时间问题。有很多方法可以使你的 client 的时间和 server 的时间进行同步。其中一个简单方法就是使用接收到的 sense_body 信息。
- 小心低速网络。如果你解决时间问题不是非常有效，那么在一个拥挤或低速的网络中，你的 client 将会有不正常表现，或者他们将会没有运行需要的资源（如：你在一台低配置机器上运行很多 client）。在这种情况下，他们可能见到旧的位置信息，然后在那个条件下进行动作，这样就会造成混乱（如：他们绕着自己转圈）。
- 标记的主要用途是帮助球员找到自己在场上的正确位置。你的第一个 client 可以忽略标记，使用系统的相对位置。但是不久后，你就会需要有一个位置模型。
- 在分析感知信息时，程序应该检查缓冲区的结尾。感知信息使用 S-expression。但是如果感知数据比缓冲区要大的话，expression 可能还没有完全结束，主要就会丢失东西。如果还用原先的方法去分析的话。可能会造成程序 core－dump。

接下去是 coach

Soccermonitor and soccerserver are connected via UDP/IP on port 6000 (default).监视器 6000

Note that the init and reconnect commands should be send to the player's UDP port (default: 6000) of the Soccer Server machine,程序 client6000

If the server is invoked with one of the trainer modes, it prepares a UDP socket to which the trainer-client can connect. The default port number is 6001训练者6001
The default port number for online coaches is 6002.在线教练6002

以下翻译自 manual7.07later

7. The coach

7.1 介绍
教练 coach 是为其他球员提供帮助的特殊 client。有两种 coach：在线教练 online coach 和训练者 trainer。后者也常被叫做离线教练，但为了更清楚的描述，我们将使用训练者 trainer 的叫法。

7.2 训练者和在线教练的区别
一般而言，训练者能对比赛实行更多的控制，仅仅在开发阶段使用；而在线教练则可以在正式比赛时使用。在自学习或者自动控制开发时，训练者是非常有用的。在线教练则是在比赛时为球员提供更多的建议和信息。

在开发球员 client 时，比如对带球、踢球技术进行机器学习时，以自动方法构造训练场景是非常有效的。因此，训练者应该有以下的功能：

- 能控制比赛模式 play-mode。
- 能广播听觉消息。消息应该可以包括一个命令或者一些其他球员的信息。它的语法和语义是用户自定义的。
- 能将球员和足球移动到场上的任何位置，同时可以设定他们的方向和速度。
- 能得到运动物体的无噪声信息

更多的细节见 7.3

在线教练则是企图观察比赛然后为球员提供建议和信息。因此，他的能力有所限制：

- 能和球员通讯
- 能得到运动对象的无噪声信息

为了防止教练集中式的控制每一个球员，通讯能力被限制了，详见 7.7 节。在线教练是一个很好的工具，可以进行对方建模，比赛分析以及给己方球员战略指导。由于教练能得到场上的无噪声全部信息，而且实时要求不高，因此教练可以花费更多的时间考虑战略。在线教练的更多细节详见 7.6 节。

7.3 Trainer 训练者

7.3.1 和带裁判的 server 连接、和不带裁判的 server 连接

默认情况下，soccerserer 内带的裁判模块被激活，控制比赛（见 4.7 节）。如果训练者想完全控制比赛，则必须告诉 soccerserer 不要激活裁判模块。也就是，举例来说，在进球后，比赛模式将不会再改变，球员也不会被移动到己方半场。训练者必须对这些事件按照自己的规则做出反应。

Soccerserver 必须在刚启动的时候就被告知要使用训练者。如果要使用 coach 而且内置的裁判功能需被关闭，则增加一个参数-coach（这个表示是 offline-caoch，和 online-coach 区别）到 server 应用程序的命令参数中。你也可以在 server.conf 文件增加一行 caoch。

如果你想连接训练者，而 server 的裁判功能还是激活的，那么增加 server 的命令行参数 –coach_w_referee。或者在 server 的配置文件中增加 coach_w_referee 行。

如果 server 使用以上方法调用的话，那么就会准备好供训练者连接的 UDP 连接。默认端口是 6001（在线教练的默认端口是 6002）。如果想使用不同的端口号，可以修改端口号参数（见 4.9.1 节）。

7.4 命令

训练者和在线教练可以使用下面的一套命令。命令分成三部分。第一部分是仅能由训练者执行的命令，第二部分则还能被在线教练有限制的执行，第三部分是训练者和在线教练都能执行。

7.4.1 仅仅能由训练者执行的命令

- **(change_mode PLAY_MODE)**

改变比赛模式至 PLAY_MODE。PLAY_MODE 必须是 4.7.1 节中定义的模式。请注意在大部分情况下，soccerserer 仅仅修改比赛模式当它接收到这个命令时。足球的位置往往是不改变的，但是在有些情况下，会移动球员位置。比如：在 free-kick、kick-in 时，站在某一圆周范围内的球员将会被移开。当切换到 before-kick-off 模式时，他们会被移到他们自己场地。

Soccerserver 会返回以下信息：

– **(ok change_mode)**

成功执行命令

– **(error illegal_mode)**

指定的比赛模式无效

– **(error illegal_command_form)**

比赛模式参数被忽略

- **(move OBJECT X Y [VDIR [VEL_X VEL_Y]])**

本命令移动一个对象，可以是一个球员也可以是足球（格式详见以上小节），到绝对位置

（X，Y）。如果命令中包括了 VDIR，将会将对象的脸的朝向也修改为 VDIR（这个近对球员起作用）。另外的，如果定义了 VELx 和 VELy，也相应的置对象速度。
可能从 server 返回的消息：

– (ok move)

命令被成功执行

– (error illegal_object_form)

命令中的对象参数无效

– (error illegal_command_form)

命令中的位置、方向或者速度无效

- (check_ball)

请求 soccerserer 返回足球位置。有以下的四种位置：

– in_field

足球在球场内

– goal_l

足球在靠近左方球门的区域

– goal_r

足球在靠近右方球门的区域

– out_of_field

足球在其他地方

注意状态 “ goal_l ” 和 “ gaol_r ” 并不表示足球已经通过了球门线。

从 soccerserver 返回的消息：

– (ok check_ball TIME BALLPOSITION)

返回 BALLPOSITION 就是上面定义的一种

- (start)

本命令启动 server，也就是说，将比赛模式设置为 “ kick_off_l ”。实质就是在 monitor 上点击 kick off 按钮。

如果 trainer 不发送初始化命令，那么从 trainer 发送出的第一个命令都会启动 server，就是将比赛模式置为 “ kic_off_l ”。

从 soccerserver 返回的消息

– (ok start)

命令被成功执行

- (recover)

本命令复位球员的体力、体力恢复率、体力使用效率和听觉能力，恢复到和刚开始比赛时相同。

从 soccerserver 返回的消息：

– (ok recover)

命令被成功执行

• (ear MODE)

此命令决定是否向 trainer 发送听觉信息。MODE 必须是 on 或者 off。如果发送了（ear on），server 将会发送所有的听觉信息给 trainer。其中的格式见表 7.3。如果发送了（ear off），server 停止向 trainer 发送听觉信息。

从 soccerserver 返回的消息：

– (ok ear on)

(ok ear off)

这两个返回消息都表示命令被成功执行

– (error illegal_mode)

MODE 不是 on 也不是 off

– (error illegal_command_form)

MODE 参赛被忽略

7.4.2 既能被 trainer 执行，也能被 online coach 限制地执行地命令

• (init (version VERSION))

trainer 执行

• (init TEAMNAME (version VERSION))

online coach 执行

这两个命令告诉 server 使用哪个版本的格式和 trainer、coach 进行通讯。对于 online coach 而言，TEAMNAME 必须指定，表明 coach 是属于哪支队伍的。请注意必须在起码一支球队已经连上 server 才运行 coach。

虽然 trainer 不需要发送初始化命令，但是建议 trainer 这样做。否则的话，server 将会使用老版本协议进行通讯。

还要提一下，trainer 的默认端口是 6001，coach 的默认端口是 6002

从 soccerserver 返回的消息：

– (init ok)

对于 trainer 而言，命令被成功执行

– (init SIDE ok)

对于在线教练而言，命令被成功执行。 SIDE 是“ l ”或者是“ r ”。

• (say MESSAGE)

在线教练也可以用相同的语法使用这个命令，但是增加了不少限制，详见 7.6.2 节。

如果是 trainer 的话，MESSAGE 将会被发送给所有的球员，如果是在线教练的话，将会发送给教练所属的球员。对 trainer 而言，MESSAGE 的格式和球员的格式是一样的。这是个字符串，而且长度要小于 say_coach_msg_size（见 4.9.1 节），字符串由数字、字母以及符号().+*/?<>_组成。

球员听到这些消息的格式见 4.3.1 节

从 soccerserver 返回的消息：

– (ok say)

命令被成功执行

– (error illegal_command_form)

MESSAGE 不符合要求的格式

- (change_player_type TEAM_NAME UNUM PLAYER_TYPE)

适用于训练者

- (change_player_type UNUM PLAYER_TYPE)

适用于在线教练

这两个命令被用来修改异构球员类型（见 4.6 节），将 TEAM_NAME 球队的 UNUM 号球员修改为 PLAYER_TYPE。PLAY_TYPE 是介于 0 到 6 之间的一个整数，0 表示是默认的球员类型。对于 online coach，没有了 TEAM_NAME 参数，因为 online coach 修改的肯定只能是己方的球员类型。

训练者无需遵守以下规定：只能更换三次（由 subs_max 指定）场上球员的类型

当在线教练更换球员时，更多的限制规则见 7.6.3 节

从 soccerserver 返回给训练者或者在线教练的消息：

– (warning no_team_found)

球队不存在

– (error illegal_command_form)

如果 change_player_type 后面没有跟随一个字符串、两个数字以及一个右括号。

– (warning no_such_player)

如果那个球队没有对应球员号码的球员

– (ok change_player_type TEAM UNUM TYPE)

命令成功执行

soccerser 还会发送给在线教练更多的以下消息：

– (warning cannot_sub_while_playon)

如果比赛模式是 “ play_on”

– (warning no_subs_left)

如果在线教练已经在本次比赛中更换三次了（由 subs_max 指定）

– (warning max_of_that_type_on_field)

如果球员类型不是默认的，而且在场上已经有三个同类型的球员了（由 subs_max 指定）？？？

– (warning cannot_change_goalie)

如果在线教练企图修改守门员的球员类型

与此同时，server 返回给本队球员的消息：

– (change_player_type UNUM TYPE)

server 发送给对方球员（包括对方在线教练）的消息：

– (change_player_type UNUM)

7.4.3 能被训练者和在线教练执行的命令

- (look)

这条命令提供下列对象的位置信息：
——左方和右方的球门
——足球
——所有活动球员
请注意训练者以及双方的在线教练接收到的是左方的坐标。也就是说，坐标系是左半场球员的坐标系。一般情况下，球员不会接收到绝对坐标信息（唯一的除了 move 命令中的参数），普遍做法都是定位自身，使本方球门 x 为负值。
从 soccerserver 返回的消息：
– (ok look TIME $(OBJ_1$ $OBJDESC_1)$ $(OBJ_2$ $OBJDESC_2)$ ...)
OBJj 可以是上述的任一对象。这些对象的名字组成由 4.3 节所示。OBJDESCj 有以下各项：
对于球门： X Y
对于足球： X Y DELTAx DELTAy
对于球员： X Y DELTAx DELTAy BODYANGLE NECKANGLE

如果训练者或者在线教练希望周期性的得到视觉信息，使用（eye on）命令
- (eye MODE)

MODE 必须是 on 或者是 off。 如果发送了（eye on），server 会每隔 100 毫秒（间隔由 send_vi_step 参数确定）自动的向 client 发送(see_global …)信息。如果(eye off)被发送，server 将不会自动的给 client 发送视觉信息。在这种情况下，如果训练者或者在线教练需要视觉信息，就发送（look）命令。
从 soccerserver 返回的消息：
– (ok eye on)
(ok eye off)
这两个都表明命令被成功执行
– (error illegal_mode)
MODE 不是 on 或者不是 off
– (error illegal_command_form)
MODE 参数被忽略

- (team_names)

这个命令向训练者或者在线教练返回两支球队队名以及他们所在的半场
从 soccerserver 返回的消息：
– (ok team_names [(team l $TEAMNAME_1$) [(team r $TEAMNAME_2$)]])
基于和 server 连接的球队数，会提供 0 个、1 个、2 个球队的名字。回忆一下，第一支连接的球队将处于左半场。

7.5 来自 server 的消息
除了上述的 server 返回消息外，还有其他消息。如果 client 和 7.0 版本极其以上的 server 连接（见 init 命令），他们将同球员一样，得到一下的参数消息：

- (server_param ...) once
- (player_param ...) once
- (player_type ...) once for each player type

更多的细节见 4.2.2 节

如果 client 发送（eye on）命令，选择每周期都得到视觉信息，它将会每隔 100ms（send_vi_step）接收到如下格式的消息：

(see_global (OBJ_1 $OBJDESC_1$) (OBJ_2 $OBJDESC_2$) ...)

OBJj 可以是上述的任一对象。这些对象的名字组成由 4.3 节所示。OBJDESCj 有以下各项：

对于球门： X Y

对于足球： X Y DELTAx DELTAy

对于球员： X Y DELTAx DELTAy BODYANGLE NECKANGLE

语法和（look）命令的返回值相同

如果 client 发送（ear on）命令（仅能 trainer 执行），它将会接收到所有的听觉信息，包括两个教练已经所有的球员。有两种听觉格式：

- (hear TIME referee MESSAGE)

表示了裁判消息，比如 “ play_on ” 和 “ free_kick_left ”。关于裁判的消息格式详见 4.7 节。

- (hear TIME (p "TEAMNAME" NUM) "MESSAGE")

所有球员的话语消息。请注意其中的引号。

关于球员的说话以及听觉能力详见 4.3.1 节

7.6 在线教练

7.6.1 简介

在线教练能够在正式比赛时作为特殊 client 和 server 连接。他可以接收到场上对象的全局的无噪声信息。为了鼓励这项研究，从 2001 年开始有专门的教练比赛。这种方式下，不想开发球队程序的研究人员也可以通过开发在线教练来加入 RoboCup 的挑战。另外，为了使教练程序能够对不同的球队兼容，已经开发出了一种标准教练语言来和球队交流。

关于在线教练和 server 相互通讯的消息格式详见 7.4 和 7.5 节。

7.6.2 和球员的通讯

对 7.00 以上版本，当比赛模式不是 “ play_on ” 时，在线教练可以发送一定长度的（128 字符，say_coach_size）的字符数字（再加上符号().+*/?<>_）消息。这些消息的格式仍旧是自由的，不过现在有了另外一种标准语言格式。

为了避免在线教练控制各个球员的每一个微小动作，通过以下方法来限制其通讯能力。在标准教练（在线教练简称，下同）语言中，共有四种类型的消息：advice，define，info 和 meta 消息。每隔 300 个周期（由 clang_win_size 指定）教练能发送其中的一个类型。该类型的允许发送数量由 clang_advice_win, clang_define_win, clang_info_win, clang_meta_win 参数确定（见 4.9.1 节）。在 50 个周期（由 clang_mess_delay 确定）以后，球员能够听到这些消息。如果比赛模式不是 “ play_on ”，每个周期都可以发送一条（由 clang_mess_per_cycle 确定）消息给球员，即使延时还没有完毕。在非 “ play_on ” 模式下发送的消息不计入消息数量的限制中。如，在使用默认值的情况下，在中断情况下教练发送的每周期一条消息能够无延时的被球员听见。Server 确保将来自 coach 的消息有序的发送给球员。

下述的语法并没有取代能发送给 server 的消息长度限制。然而，为了特别的原因，标准语

言中任何消息的长度都不会超过 2013 字节（所以发送给球员的最长长度将是 2048 字节？）。

对于无格式消息，教练仅仅能在非“ play_on ”模式下说话。在每场比赛中，教练可以发送 say_coach_cnt_max 条无格式消息。这些消息的长度都小于 say_coach_msg_size。如果比赛进入加时模式，每 6000 时钟周期，在线教练都会被增加 say_coach_cnt_max 条消息的能力。被允许的消息数量是可以积累的，所以如果教练没有用完，可以在加时赛中继续使用。当教练到达消息的上限值后，server 会返回（error said_too_many_messages）。

标准教练语言见 7.7 节细述。

7.6.3　在比赛时改变球员类型

使用 change_player_type 命令（见 7.4 节所述），在线教练可以在“ before_kick_off ”时无限制的改变球员类型。当然这些改变要符合异构球员的一般规则（见 4.6 节）。比赛开始后，在非“ play_on”模式中还可以改变三次。

关于从 server 返回消息详见 7.4 节。

提示：server 返回给本队球员的消息：

– (change_player_type UNUM TYPE)

server 发送给对方球员（包括对方在线教练）的消息：

– (change_player_type UNUM)

7.7 标准教练语言

7.7.1 基本属性

标准教练语言是为了和不同球队程序进行兼容而开发的。设计目的之一是为了定义清晰的语义，避免球员和教练间的误解。语言基于底层概念，能够在高层概念中结合使用。

另外，教练可以通讯一定数量的自由消息，格式任意，在非“ play_on ”模式下发送给球员。详见 7.6.2 节。当然，如果你计划为其他球队配备你的教练程序，很有可能其他队伍不明白教练格式。

下面的教练语言是我们开发的第一版。我们希望有更多的人来参与开发。

提醒，server 本身使用 flex 和 bison（GNU 下面的 lex 和 yacc 版本）来分析所有的教练语言，也使用了基于 c＋＋类的一个简单结构。处理分析这些语言是被授权的。细节方面，请见 coach_lang.[Chly]和 coach_lang_comp.[Ch]文件。

7.7.2 五种类型的消息简述

在标准教练语言中，一共有五种消息格式：Info, Advice, Define, Meta 以及 Freeform 即无格式。下面的我们还用不到，就不翻了，呵呵。以后是一定需要用到的，以后在翻译。